Synthesizing and Optimizing FDIR Recovery Strategies From Fault Trees

Liana Mikaelyan[1][0000−0002−2788−3777], Sascha Müller[1][0000−0002−1913−1719], Andreas Gerndt[1][0000−0002−0409−8573], and Thomas Noll[2][0000−0002−1865−1798]

[1] Software for Space Systems and Interactive Visualization,
DLR (German Aerospace Center), 38108 Braunschweig, Germany
{Liana.Mikaelyan, Sa.Mueller, Andreas.Gerndt}@dlr.de
[2] Software Modeling and Verification Group,
RWTH Aachen University, 52056 Aachen, Germany
Noll@cs.rwth-aachen.de

Abstract. Redundancy concepts are an integral part of the design of space systems. Deciding when to activate which redundancy and which component should be replaced can be a difficult task. In this paper, we refine a methodology where recovery strategies are synthesized from a model of non-deterministic dynamic fault trees. The synthesis is performed by transforming non-deterministic dynamic fault trees into Markov Automata. From the optimized scheduler, an optimal recovery strategy can then be derived and represented by a model we call Recovery Automaton. We discuss techniques on how this Recovery Automaton can be further optimized to contain fewer states and transitions and show the effectiveness of our approach on two case studies.

Keywords: FDIR · Fault Tree Analysis · Synthesis · Formal Methods.

1 Introduction

Reliability engineering is an important discipline in the design of any safety critical system, in particular in the domain of aerospace systems and spacecraft. No matter how well designed a system is, it still has to deal with the presence of faults to some extent. Faults in this context can be events such as equipment failure, wrong sensor readings, external interferences and many more. To raise trust in handling system failures, reliability engineering tries to embed Failure Detection, Isolation and Recovery (FDIR) concepts. These concepts are derived using various tools and methodologies such as Fault Tree Analysis (FTA) [9].

FTA is a methodology commonly used in the industry for performing state-of-the-art failure analysis [13]. The resulting Fault Trees (FT) describe how faults propagate through components and subsystems of a system and eventually lead to a top-level system failure. Graphical representations of these trees are intuitive and easy to understand. On the one hand, FTs can be used to analyze the system qualitatively in terms of fault combinations that lead to system failure. On the other hand, they also enable quantitative analysis of

important computable measures such as reliability. Dynamic Fault Trees (DFT) are an extension introducing temporal dependencies and new features to analyze redundancy concepts known as spare management. However, there are challenges arising from non-deterministic behavior of DFTs such as spare races. An example for such race behavior can be seen in a system of two operative memories together with a pool of two spare memories. If both operative memories fail at the same time it is unclear which backup memory takes over the role of which operational one.

To overcome this shortcoming, a new methodology was presented in [11]. It introduces a model of Non-deterministic Dynamic Fault Trees (NdDFT) as an extension to DFTs. In contrast to the latter, the new NdDFT does not impose a fixed, rigid order on the spares to be used. As next step, the methodology foresees transforming this NdDFT model into a Markov Automaton (MA) which is suitable for the computation of the aforementioned non-deterministic decisions on spare activations. By optimizing the scheduling of the MA model in terms of reliability of the system, a recovery strategy for the NdDFT can be synthesized. This recovery strategy defines which spare has to be used in which failure state of the system and can therefore guarantee an optimal reliability at all times.

The goal of the present paper is to refine the methodology presented in [11] by further developing an automata model that formalizes the decision process underlying a recovery strategy, a so-called Recovery Automaton (RA). We give its formal definition and show how it can be minimized in order to obtain an efficient implementation of recovery strategies for FDIR.

This paper is structured as follows. Section 2 of this paper summarizes the related work relevant to the topic of FTs, MA and synthesis of recovery strategies. Further background on the theory of FTs including their (non-deterministic) dynamic variants is given in Section 3. Section 4 describes the process of synthesizing recovery strategies from a given NdDFT as well as a model to represent such strategies, which is further optimized in Section 5. Section 6 then evaluates the technique on a use case example. Finally, the paper concludes in Section 7 and provides some outlook to future work.

2 Related Work

The goal of FDIR lies in keeping a system in a stable and operational state, even in the presence of faults. While some of the following steps may be omitted in some cases, performing FDIR generally means applying the following procedural approach [15]:

- Monitor the system to detect the occurrence of faults.
- Identify the fault and localize it within the system.
- Isolate the fault and prevent further propagation into other parts of the system.
- Perform recovery actions to reconfigure the system and return it into a stable state.

In order to derive how faults relate to each other and eventually lead to a system wide failure, failure analysis techniques such as FTA can be employed. One of the very basic types of FTs are Static Fault Trees (SFT). They employ Boolean algebra to combine various different failure events by AND and OR operations, often graphically represented as gates, until they sum up to the overall system failure. The failure events are usually related to faulty components of the system. Applying this methodology, statements such as "The system fails if component A and component B fail" can be modeled and refined to arbitrary levels of precision. The probability of the top-level failure after time t (reliability) can be computed from a given DFT for example by transforming a DFT into a Continuous-Time Markov Chain (CTMC) [5].

Markov Automata [6] are an extension to CTMCs. They are state-based transition systems with two types of transitions: They can contain continuous-time transitions (also called Markovian transitions) that are labeled with rates, that is, non-negative real values as well as immediate, non-deterministic transitions labeled by actions. In the latter case, transitions have to be chosen by a so-called scheduler. The computation of optimal schedulers for Markov Automata with respect to various quantitative objectives, such as state reachability, is discussed in [7].

Computing strategies for recovery purposes from a given fault model has been researched in other contexts. In [1], a similar approach is taken for repairable fault trees. The authors consider non-deterministic repair policies where the repair order is not fixed. Optimal repair policies are then computed by converting the repairable fault tree to a Markov decision process, a time-discrete version of Markov Automata. However, the authors do not consider DFT models. In [4], Dynamic Decision Networks (DDN) are employed and their inference capabilities are exploited to create autonomous on-board FDIR systems for spacecraft that can select reactive and preventive recovery actions during run-time. In [12], the authors propose creating the DDN from an extension of the DFT model. Timed Failure Propagation Graphs are used in [2] to synthesize FDIR components, namely monitors for the purpose of fault detection and recovery plans for every specified combination of fault and mode. Here, the recovery components are created using a planning based approach on predefined actions.

3 Fault Trees

FTs are graphs consisting of two types of nodes respectively representing events and gates. The root node, or top level event (TLE), usually represents the event of a system failure whereas the leaves of the tree model the event of individual components failing. The leaves are also called basic events (BE). They correspond to a Boolean variable where false represents the initial state of no failure. The variable is considered true in case of a failure event. We consider here only the case of permanent failure, i.e. once a BE has failed, it remains in a failed state for all future points in time. The branches of the trees are represented by the gates performing operations on the events. FTs are directed acyclic graphs

(a) BE (b) Non-basic event (c) OR (d) AND (e) k-VOTE

Fig. 1. Gates and events in a Static Fault Tree

starting from the BEs pointing over the gates towards the system failure event. In the following, basic events will be denoted by $b_1, b_2, \ldots$, sets of basic events by $B_1, B_2, \ldots$ and failure rates by $\lambda_1, \lambda_2, \ldots$.

3.1 Static Fault Trees

Fig. 1 shows the gates and events used in the SFT notation. SFTs use Boolean operations represented by AND and OR gates. There also exist other gates such as the k-VOTE gate, which propagates if at least k inputs have failed. Observe that a 1-VOTE gate corresponds to an OR gate and a k-VOTE gate with k inputs to an AND gate. Implementation wise, all gates can therefore be considered as k-VOTE gates for some appropriate k. Some other extensions also introduce a NOT gate. However, this allows the construction of fault trees where the TLE can change from having failed to working again as new failures occur. These fault trees are known as non-coherent fault trees and have been dismissed as being a sign for modeling errors [14].

3.2 Dynamic Fault Trees

Many extensions have been proposed to the formalism of FTs [13] to increase its expressiveness and enhance its features. A particular extension is the notion of Dynamic Fault Trees (DFT). It introduces temporal understanding and new features to analyze redundancy concepts known as spare management. In DFTs, a node can be either failed, active (operational) or dormant (operational). A node that is an unactivated spare is dormant. All other nodes are activated. Together with this state, failure rates for failing actively and failing dormantly can be defined for every BE.

Fig. 2 depicts the notation to extend SFTs to DFTs introducing new gates POR, PAND, SPARE and FDEP. The PAND (priority AND) gate propagates in case all inputs fail exactly in sequence from left to right. The POR (priority OR) gate propagates in case the leftmost input occurs before all other inputs.

(a) POR (b) PAND (c) SPARE (d) FDEP

Fig. 2. Standard dynamic gates

The SPARE gate is connected to a primary event and a set of spare events. It propagates a failure if the primary input failed and all spares are either claimed or failed themselves. The spare events can be shared with another SPARE gate, therefore a spare can be claimed by either the one or the other SPARE gate. But there may be no shared elements between the primary input and any spare. The order in which such a spare is chosen is deterministic and defined at design time by the reliability engineer.

The FDEP (functional dependency) has a trigger event on the left hand side and any number of dependent events functionally dependent on the triggering event. When the trigger event occurs, the dependent events are set to fail as well. The output of an FDEP gate only indicates to which tree it belongs and has no further semantical meaning.

In the following, we give an example to illustrate the DFT notation. Fig. 3 shows a system consisting of two memory components which are covered by two spare memories for failures. The two spares are shared among the two SPARE gates. According to DFT semantics, Memory3 will be used before Memory4 in case of a failure of Memory1 or Memory2. In addition, the system has two hot redundant, always active power sources, Power1 and Power2. Both primaries Memory1 and Memory2 are powered by Power1 and the redundancies Memory3 and Memory4 are powered by the second power source Power2. Using FDEPs, the failure of a power source is propagated to the respective memory components. In the figure, FDEP dependent events are marked by an arrow and dashed lines indicate the parent of an FDEP.

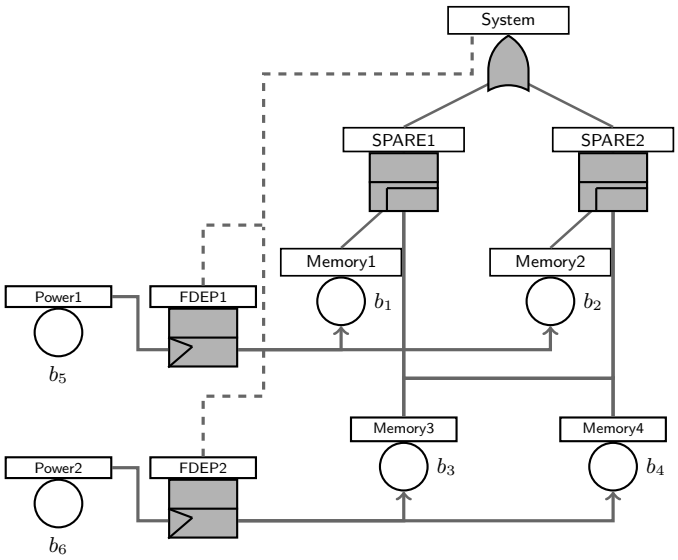


Fig. 3. Example DFT

3.3 Non-Deterministic Dynamic Fault Trees

As described before, DFTs require spares to be activated in a fixed and rigid order. This order cannot be adapted depending on faults that have previously occurred. Additionally, in cases of spare races it is not semantically clear which SPARE gate claims the actual redundancy. To relax on this semantical restriction of the DFT model, [11] introduces an inherently non-deterministic DFT model (NdDFT, following the naming in [1]). The syntax and notation of the NdDFT is completely adopted from the DFT. Semantically, the NdDFT drops the requirement that spares are always activated from left to right. Morover, the new non-deterministic semantics allows for a SPARE gate to leave the spares available for more important SPARE gates by not claiming. Whenever BEs occur in an NdDFT, the new semantics allow to perform valid recovery actions of the following form:

Definition 1 (Recovery Action). *A recovery action r in an NdDFT $\mathcal{T}$ is an action of the form*

- $[]$ *(empty action) or*
- $CLAIM(G, S)$ *(spare gate G claims spare S, where S is a spare of G).*

We denote the set of all recovery actions possible in an NdDFT $\mathcal{T}$ by $R(\mathcal{T})$ and the set of recovery action sequences by $RS(\mathcal{T}) := (R(\mathcal{T}) \setminus \{[]\})^*$. Similarly we denote the set of all non-empty subsets of basic events by $BES(\mathcal{T})$.

4 Synthesizing Recovery Strategies

Here we describe the essential steps; details can be found in [11]. First, the NdDFT model is transformed into a Markov Automaton (MA) that represents all possible (non-deterministic) decisions on spare activations. By optimizing the scheduling of the MA model in terms of reliability of the system, a recovery strategy for the NdDFT can be synthesized. This strategy is represented by a Recovery Automaton (RA) that defines which spare has to be used in which failure state of the system and can therefore guarantee an optimal reliability at all times. The latter can be computed by a quantitative analysis of the Markov Chain that is obtained from the RA, enriched by the failure rates of basic events as determined by the original NdDFT. Fig. 4 visualizes the procedure.

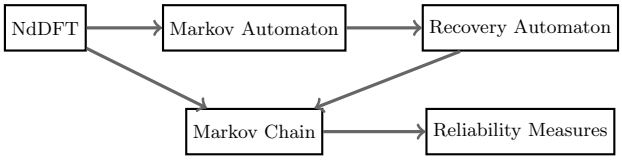


Fig. 4. Transformation road map

4.1 Recovery Strategies and Automata

In the NdDFT, the actual recovery action r that is applied is defined by a given recovery strategy. In the following, transitions of Recovery Automata are labeled by recovery action sequences. Given the observed basic events, a recovery strategy is then a mapping that returns the recovery action sequence that should be taken accordingly. The NdDFT considers recovery strategies that only have recovery actions as given in Def. 1. They are defined as follows:

Definition 2 (Recovery Strategy). *A recovery strategy for an NdDFT $\mathcal{T}$ is a mapping $Recovery : BES(\mathcal{T})^* \to RS(\mathcal{T})^*$ such that*

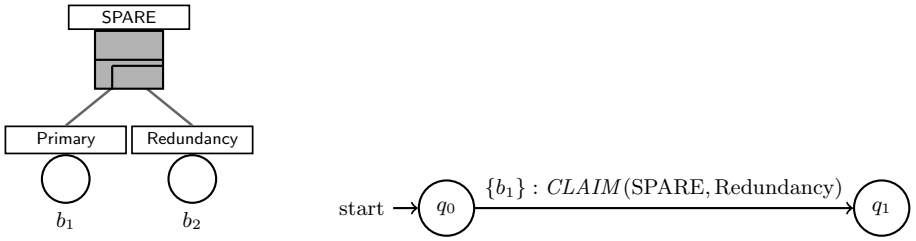
- $Recovery(\varepsilon) = \varepsilon$ *and*
- $Recovery(B_1, \ldots, B_n) = Recovery(B_1, \ldots, B_{n-1}), rs_n$ *with* $rs_n \in RS(\mathcal{T})$.

As each basic event can occur at most once, the recovery strategy only needs to be defined for pairwise disjoint sets of basic events, i.e., $B_i \cap B_j = \emptyset$ for $i \neq j$. A finite automaton that represents a recovery strategy will be called Recovery Automaton.

Definition 3 (Recovery Automaton). *A Recovery Automaton (RA) $\mathcal{R}_\mathcal{T} = (Q, \delta, q_0)$ of an NdDFT $\mathcal{T}$ is an automaton where*

- *Q is a finite set of states,*
- *$q_0 \in Q$ is an initial state, and*
- *$\delta \colon Q \times BES(\mathcal{T}) \to Q \times RS(\mathcal{T})$ is a deterministic transition function that maps the current state and an observed set of faults to the successor state and a recovery action sequence.*

The recovery strategy induced by a Recovery Automaton $\mathcal{R}$ is denoted by $Recovery_\mathcal{R}$. An example of a Recovery Automaton for a simple Fault Tree consisting of a SPARE gate with a cold redundant spare is given in Fig. 5.



(a) NdDFT (b) Recovery Automaton

Fig. 5. Example of (a) NdDFT and (b) RA

4.2 Non-Deterministic Dynamic Fault Trees to Markov Automata

Transforming an NdDFT to a Markov Automaton can be done by adapting traditional algorithms for transforming DFTs to CTMCs. As base algorithm, we use the one given in [5]. The adapted algorithm operates by memorizing two sets of data in every of its states: First, the history of occurred basic event sets $(B_1, B_2, \ldots, B_n)$. Second, a mapping from spare gates to the currently claimed spare. The initial, empty history of the algorithm is denoted by $()$. Starting with this initial state, all active basic events, i.e. those that are not associated to an unactivated spare, are used to compute Markovian successors for each of them while extending the history accordingly.

The respective basic event set is obtained by taking the active basic event and computing all basic events that transitively fail due to FDEPs. The transitions are labeled with the respective failure rate of the basic event causing the transition. All transitions that would lead to a state that implies that the top-level event (system failure) has occurred, are connected to a special FAIL state instead. For each target state of a Markovian transition, the algorithm generates successors using non-deterministic transitions. Each non-deterministic transition is labeled by a valid recovery action.

4.3 Synthesizing Recovery Automata from Markov Automata

Using existing techniques for optimizing the scheduling of a Markov Automaton, the optimal non-deterministic transitions for maximizing the system reliability can be computed. The Recovery Automaton model is then used to represent the underlying decision process of the scheduler.

Extracting a Recovery Automaton from a scheduler for a Markov Automaton is achieved by replacing sequences of transitions for states $s_0, s_1, \ldots, s_n$ of the form $(s_0, B : \lambda, s_1), (s_1, r_1, s_2), \ldots, (s_{n-1}, r_n, s_n)$, where B is a basic event set, λ a failure rate and $r_1, \ldots, r_n$ recovery actions, by the transition $\delta(s_0, B) = (s_n, r_1 \ldots r_n)$ where empty recovery actions are ignored. This applies to all transitions where $s_1, \ldots, s_n$ are the successors computed by the optimized schedule of the Markov Automaton. All other non-deterministic transitions are then discarded. Finally, the algorithm discards all unreachable states.

5 Further Optimization of Recovery Automata

Complex systems usually exhibit a large number of faults that may occur. This means that NdDFTs describing such systems may be very large and correspondingly synthesized Recovery Automata may contain redundant states. In this section, we refine the given synthesis procedure by discussing some techniques for reducing the state space and the transition count of a synthesized Recovery Automaton. This leads to the task of finding an automaton with the same "behavior" that contains a smaller number of states. To capture this notion of having the same behavior, we introduce the concept of recovery equivalence between Recovery Automata as follows:

Definition 4 (RA Recovery Equivalence). *Let $\mathcal{R}_1 = (Q_1, \delta_1, q_{0_1})$ and $\mathcal{R}_2 = (Q_2, \delta_2, q_{0_2})$ be two RAs. We define a binary relation $\approx_R$ such that it holds true for any two RA that $\mathcal{R}_1 \approx_R \mathcal{R}_2$ iff for any sequence of sets of basic events $B_1, \ldots, B_n$ with $B_i \cap B_j = \emptyset$ for any $i \neq j$ it holds that:*

$$Recovery_{\mathcal{R}_1}(B_1, \ldots, B_n) = Recovery_{\mathcal{R}_2}(B_1, \ldots, B_n)$$

Given a Recovery Automaton as an input, the task of minimization involves obtaining an equivalent recovery automaton with as few states as possible. The standard problem of automata minimization is well-known and has been studied extensively. In this work, we apply the usual definition of trace equivalence and lift it to states of Recovery Automata:

Definition 5 (Trace Equivalence). *Let $\mathcal{R}_\mathcal{T} = (Q, \delta, q_0)$ be an RA. A trace equivalence $\approx \subseteq Q \times Q$ is a maximal, binary relation such that it holds for any states $q_1, q_2 \in Q$ that $q_1 \approx q_2$ iff for any $B \in BES(\mathcal{T})$ it holds that:*

$$\delta(q_1, B) = (q_1', rs_1) \text{ and } \delta(q_2, B) = (q_2', rs_2) \text{ with } q_1' \approx q_2' \text{ and } rs_1 = rs_2$$

Equivalent states in automata can be computed using the Partition Refinement algorithm [8] and then a minimized automaton can be obtained by merging all equivalent states. In the setting of Recovery Automata, we can go even further and merge pairs of states that are not trace equivalent as long as the behavior of the automaton does not change. A simple example for a case where merging non-equivalent states yields a Recovery Automaton that induces an equivalent recovery strategy, can be seen in Fig. 6.

In the following we present the main contribution of this work: Rules that allow to merge states that are not trace-equivalent, yet yield implementations of equivalent recovery strategies. We identified two cases where merging non-equivalent states does not change the induced recovery strategy.

- **Case 1: Merging Orthogonal States.**
- **Case 2: Merging the FAIL state to Predecessors.**

In both cases, the key to minimization that we exploit, is the fact that the inputs of the automaton are produced by an FT. Hence, basic events can only occur at most once. This leads to the effect that certain traces in the RA are not valid inputs for the correspondingly induced recovery strategy. Therefore it gives us additional freedom for merging states that do would not be allowed to be merged in a standard automaton model.

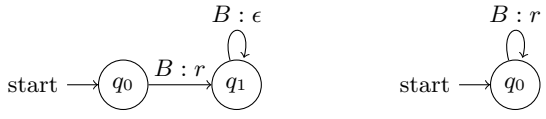


Fig. 6. (a) initial RA; (b) minimized RA

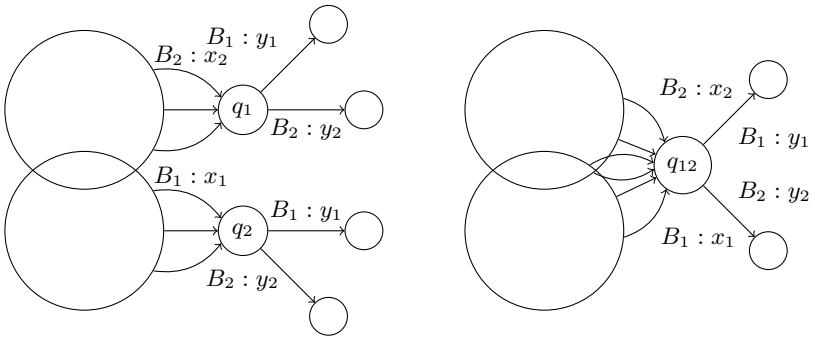


Fig. 7. (a) initial RA; (b) RA after merging states q_1 and q_2

5.1 Merging Orthogonal States

In the first rule, the idea is to identify states that may have transitions with disagreeing outputs, but where we can guarantee for certain that those transitions can never be taken, as their necessary inputs can no longer be produced. As mentioned before, the key to this idea lies in the exploitation of the property that basic events can only occur at most once in an FT. This gives us the following observation: If a basic event occurs on every path leading to a state in an RA, then it is guaranteed that in the future no transition listing this basic event in its guards can be taken. Note that Recovery Automata are deterministic automata, meaning that unlike non-deterministic automata they always have a transition defined for every possible input. Fig. 7 abstractly illustrates the application of this merging rule.

For the purpose of formalizing the intuitively given notion, we now introduce the concept of *orthogonal states*. To capture the basic event sets that can no longer be produced by an FT upon having reached a state in the RA, we define the set of guaranteed inputs of a state q as a function $GI\colon Q \to Q$ with:

$$GI(q) := \{B \in BES(\mathcal{T}) \mid \text{for all paths } q_0 B_0 : rs_0 \dots q_{n-1} B_{n-1} : rs_{n-1} q \\ \exists i : B_i \cap B \neq \emptyset\}$$

In order to compute the set of guaranteed inputs, we apply the work list algorithm [10] using the following transfer functions:

$$GI(q_0) := \emptyset$$
$$GI(q) := \bigcap_{(p,B) \in pred(q)} GI(p) \cup \{B\}$$

With $pred(q) := \{(p, B) \mid \delta(p, B) = (q, rs) \text{ for some } rs, p \neq q\}$ denoting the set of predecessor transitions of a state q. Having setup these preliminary definitions, the concept of orthogonality between states can now be formalized with the following definition:

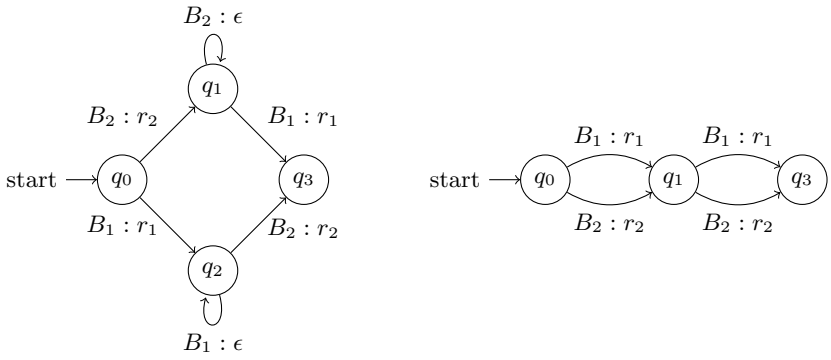


Fig. 8. (a) initial RA $\mathcal{R}_1$; (b) RA $\mathcal{R}_2$ after merging states q_1 and q_2

Definition 6 (Orthogonal States). *Let $\mathcal{R}_\mathcal{T} = (Q, \delta, q_0)$ be an RA. Let further $p, q \in Q$ be two non-initial distinct states and $B \in BES(\mathcal{T})$. Then p, q are orthogonal with respect to B iff*

$$B \in GI(p) \cup GI(q)$$

To illustrate the definition of orthogonality, we consider as an example the Recovery Automaton depicted in Fig. 8. The RA we consider there reacts to two distinct basic event sets B_1 and B_2 and performs a corresponding recovery action r_1 or r_2 accordingly. An NdDFT that would produce such an RA would be for example a system consisting of two parallel spare gates running independently from each other, e.g. spare gates with no shared spare. For the guaranteed inputs we have:

- $GI(q_0) = \emptyset$,
- $GI(q_1) = GI(q_0) \cup \{B_2\} = \{B_2\}$,
- $GI(q_2) = GI(q_0) \cup \{B_1\} = \{B_1\}$ and
- $GI(q_3) = (GI(q_1) \cup \{B_1\}) \cap (GI(q_2) \cup \{B_2\}) = \{B_1, B_2\}$.

Thus, by Def. 6 it holds that q_1 and q_2 are orthogonal with respect to basic event sets B_1 and B_2. Observe that q_1 has an outgoing loop transition labeled with $B_2 : \epsilon$ that cannot occur. Similarly, q_2 has an outgoing loop transition labeled by $B_1 : \epsilon$ that cannot occur. In the merged RA, these transitions are eliminated and all the other incoming and outgoing transitions are redirected to start and end at the merged state respectively.

We are now ready to incorporate the orthogonality concept into an equivalence definition. We extend the basic trace equivalence definition as follows:

Definition 7 (RA State Recovery Equivalence). *Let $\mathcal{R}_\mathcal{T} = (Q, \delta, q_0)$ be an RA. A state-based recovery equivalence $\approx_R \subseteq Q \times Q$ is a maximal relation such that it holds for any states $q_1, q_2 \in Q$ that $q_1 \approx_R q_2$ iff for any $B \in BES(\mathcal{T})$ it holds that either:*

- $\delta(q_1, B) = (q_1', rs_1)$ *and* $\delta(q_2, B) = (q_2', rs_2)$ *with* $q_1' \approx q_2'$ *and* $rs_1 = rs_2$ *or*
- q_1, q_2 *are orthogonal with respect to* B.

We now prove the correctness of our approach. The following theorem states that merging two recovery equivalent states yields a recovery equivalent RA.

Theorem 1. *Let* $\mathcal{R}_1 = (Q_1, \delta_1, q_{01})$ *be an RA with a pair of states* q_1 *and* q_2 *such that* $q_1 \approx_R q_2$. *Let further* $\mathcal{R}_2 = (Q_2, \delta_2, q_{02})$ *be an RA that contains equal states and transitions as* $\mathcal{R}_1$, *apart from merging* q_1 *and* q_2 *into a single state* q_{12}, *redirecting the incoming transitions of* q_1 *and* q_2 *to* q_{12} *and copying the outgoing transitions from* q_1 *with guard* $B \notin GI(q_1)$ *and* q_2 *with guard* $B \notin GI(q_2)$. *Then* $\mathcal{R}_1 \approx_R \mathcal{R}_2$.

Proof. Let $\beta := B_1, \ldots, B_n \in BES(\mathcal{T})^*$ be a sequence of basic event sets produced by an NdDFT. Then $B_i \cap B_j = \emptyset$ for any $i \neq j$. We distinguish two cases:

- Assume $\mathcal{R}_1$ never vists q_1 or q_2. By definiton of $\mathcal{R}_2$ we then have that also $\mathcal{R}_2$ does not visit q_{12}. And by definition of $\mathcal{R}_2$ again we thus immediately have that $Recovery_{\mathcal{R}_1}(\beta) = Recovery_{\mathcal{R}_2}(\beta)$.
- Assume $\mathcal{R}_1$ visits q_1 (the case of visiting q_2 is analog) upon reading B_i for some $i < n$. Now consider B_{i+1}. Let q_1', q_{12}' and rs_1, rs_{12} be such that:

$$\delta_1(q_1, B_{i+1}) = (q_1', rs_1) \text{ and}$$
$$\delta_2(q_{12}, B_{i+1}) = (q_{12}', rs_{12}).$$

  By Def. 7 this means that we have either:

  - $rs_1 = rs_{12}$ and $q_1' \approx q_{12}'$. By correctness of merging trace equivalent states we hence obtain $Recovery_{\mathcal{R}_1}(\beta) = Recovery_{\mathcal{R}_2}(\beta)$.
  - q_1, q_2 are orthogonal with respect to B_{i+1}. Then by Def. 6 it holds that:

$$B_{i+1} \in GI(q_1) \cup GI(q_2)$$

    If $B_{i+1} \in GI(q_1)$ then there exists by construction of GI an index $j < i + 1$ such that $B_{i+1} \cap B_j \neq \emptyset$. Contradiction to the definition of β. Therefore we obtain conclude $B_{i+1} \in GI(q_2)$. By construction of $\mathcal{R}_2$ this implies that the transition of q_2 is not copied and the transition of q_1 is chosen instead. Thus, $rs_1 = rs_{12}$ and $q_1' = q_{12}'$. Hence we can conclude $Recovery_{\mathcal{R}_1}(\beta) = Recovery_{\mathcal{R}_2}(\beta)$.

  In all cases we have $Recovery_{\mathcal{R}_1}(\beta) = Recovery_{\mathcal{R}_2}(\beta)$ and thus $\mathcal{R}_1 \approx_R \mathcal{R}_2$ by Def. 4.

□

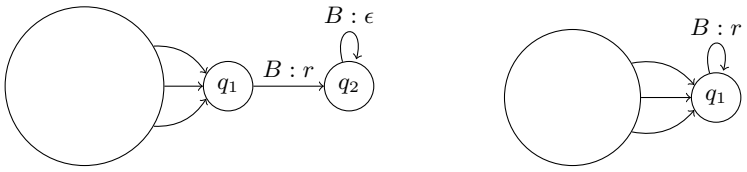


Fig. 9. (a) initial RA; (b) RA after merging FAIL states to predecessors

5.2 Merging the FAIL State to Predecessors

The idea of the second case is to identify FAIL states that do not contribute to new recovery actions sequences when a set of faults occurs. If a state only leads to a FAIL state, the transition can be turned into a self-loop. And should the FAIL state no longer be reachable, it can be eliminated. This rule is abstractly illustrated in Fig. 9. We further introduce the concept of a FAIL state.

Definition 8 (FAIL State). *Let $\mathcal{R}_\mathcal{T} = (Q, \delta, q_0)$ be an RA and $q \in Q$ a state. Then q is a FAIL state iff for any $B \in BES(\mathcal{T})$, all transitions from q are of the form $\delta(q, B) = (q, \epsilon)$.*

The formalized merging rule can then be captured by the following theorem:

Theorem 2. *Let $\mathcal{R}_1 = (Q_1, \delta_1, q_{01})$ be an RA with a pair of states q_1 and q_2 such that q_2 is a FAIL state and all transitions of q_1 are ϵ-loops except for one transition being of the form $\delta_1(q_1, B) = (q_2, rs)$, such that $rs \neq \epsilon$. Let further $\mathcal{R}_2 = (Q_2, \delta_2, q_{02})$ be an RA with equal states and transitions as $\mathcal{R}_1$, except for turning outgoing transitions of q_1 into loop transitions. Then $\mathcal{R}_1 \approx_R \mathcal{R}_2$.*

Proof. Let $\beta := B_1, \ldots, B_n \in BES(\mathcal{T})^*$ be a sequence of basic event sets with $B_i \cap B_j = \emptyset$. We distinguish two cases:

- Assume $\mathcal{R}_1$ never visits q_1. Then by definition of $\mathcal{R}_2$, it also never visits q_1. As both automata are defined to be equal otherwise, we then immediately have that $Recovery_{\mathcal{R}_1}(\beta) = Recovery_{\mathcal{R}_2}(\beta)$.
- Assume $\mathcal{R}_1$ visits q_1 upon reading B_i for some $i < n$. Then by definition, $\mathcal{R}_2$ also visits q_1 upon reading B_i. Now consider B_{i+1}. By the construction of $\mathcal{R}_2$ it holds that $\delta_1(q_1, B_{i+1}) = (q_2, rs)$ and $\delta_2(q_1, B_{i+1}) = (q_1, rs)$. for some recovery action sequence rs. Since q_2 is a FAIL state we obtain from Def. 8 that $\delta_1(q_2, B_j) = (q_2, \epsilon)$ for any $j > i + 1$. Moreover, since also $B_j \cap B_{i+1} = \emptyset$ for any $j > i + 1$ we also have by definition of q_1 and $\mathcal{R}_2$ that $\delta_2(q_1, B_j) = (q_1, \epsilon)$. In total, we can therefore conclude that:

$$\begin{aligned} Recovery_{\mathcal{R}_1}(\beta) &= Recovery_{\mathcal{R}_1}(B_1, \ldots, B_i, B_{i+1}) \\ &= Recovery_{\mathcal{R}_2}(B_1, \ldots, B_i, B_{i+1}) \\ &= Recovery_{\mathcal{R}_2}(\beta) \end{aligned}$$

In all cases $Recovery_{\mathcal{R}_1}(\beta) = Recovery_{\mathcal{R}_2}(\beta)$. Hence, $\mathcal{R}_1 \approx_R \mathcal{R}_2$ by Def. 4. □

6 Case Studies

In order to evaluate the presented techniques, we apply the synthesis methodology including the newly described merging rules to further optimize the created RA models to two use cases.

6.1 Multiprocessor Computing System

Target System. We consider the literature example of a Multiprocessor Computing System (MCS) based on the model given in [3]. The MCS consists of two main components: The Bus and the Computing Module (CM). The CM is hot redundant and consists of two further CMs CM_1 and CM_2. Each of these CMs requires a disk, a processor and a memory unit. Each CM has a warm redundant backup disk. Furthermore, a shared redundant memory unit MS is available to the entire CM in case that their own memory unit fails. Finally, both processors are powered by a common power source PS. The common power source itself is again hot redundant and consists of the two power units PS_1 and PS_2. Fig. 10 shows a NdDFT that describes the MCS.

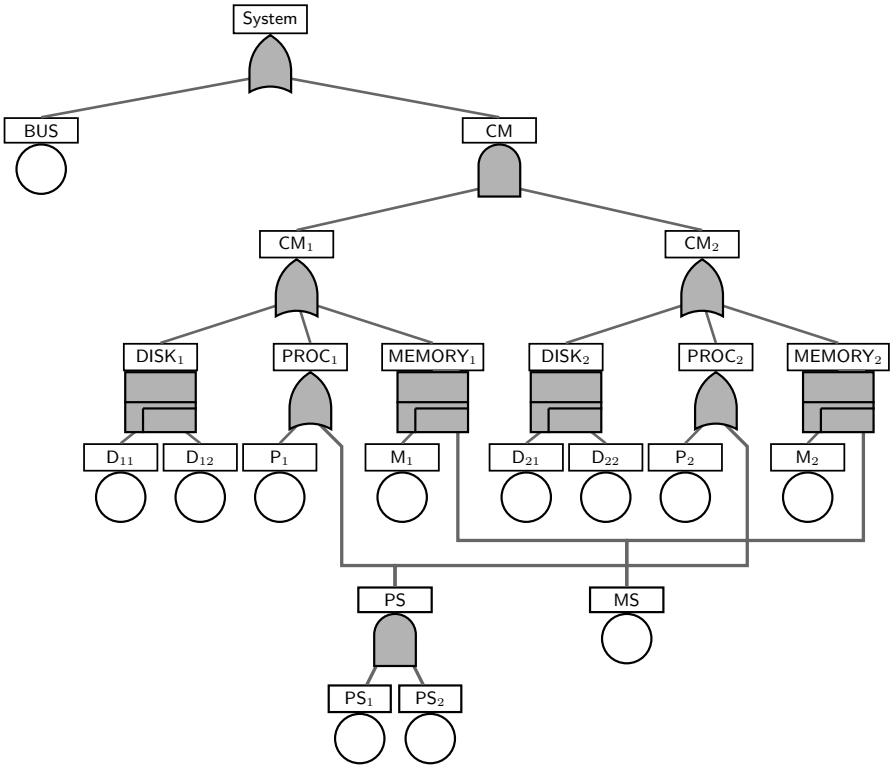


Fig. 10. NdDFT of the Multiprocessor Computing System

Experimental Results. The described synthesis algorithm was performed to obtain a Recovery Automaton from the described NdDFT. The RA was then optimized by merging trace-equivalent and recovery equivalent states and by eliminating redundant transitions.

Table 1 shows the results after minimizing the synthesized RA. Observe that initially the RA contained a large number of states and transitions. After performing the Partition Refinement algorithm based on the trace-equivalence definition, the number of states and transitions was significantly reduced. After performing the Partition Refinement and merging non-trace equivalent states according to the described merging rules, it was observed that the number of states was further reduced by 95.86% and the number of transitions was further reduced by 95.05%. Thus, merging non-trace equivalent states additionally reduced the number of states obtained by merging trace-equivalent states by 38.81% and the number of transitions was reduced by 32.38%. This indicates the effectiveness of the proposed approach to consider cases when non-trace equivalent states can be merged to obtain an equivalent Recovery Automaton having the same behavior.

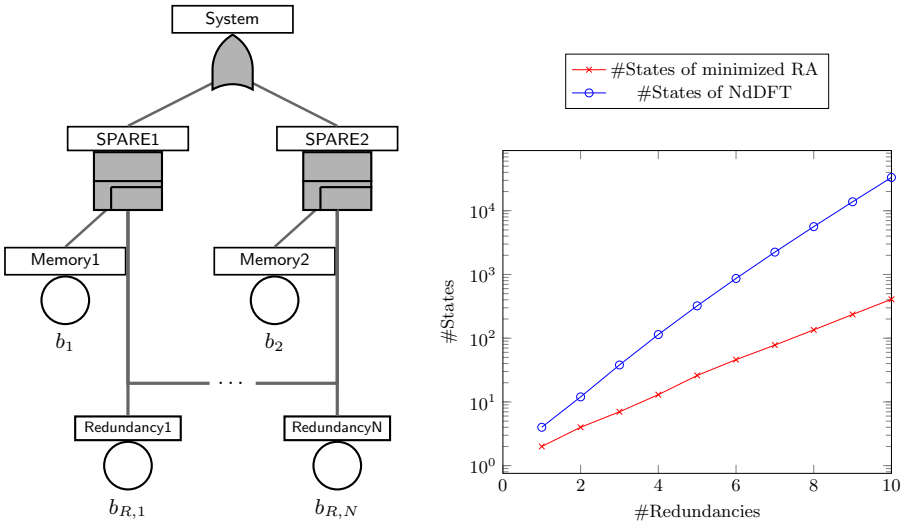
Table 1. Synthesizing and Minimizing Results.

Equivalence Relation	#States	#Transitions	States Removed	Transitions Removed
–	991	7635	–	–
Trace Equivalence	67	559	93.24%	92.68%
Recovery Equivalence	41	378	95.86%	95.05%

6.2 Memory System witn N Redundancies

Target System. To assess the the state space reduction for Recovery Automata in terms of increasing DFT complexity, we consider a family of DFTs based on the previous memory system use case given in Fig. 3. The model family is depicted in Fig. 11a. As before, the system consists of two main memory units Memory1 and Memory2. However, instead of a fixed size of redundant memory systems, they now share a variable pool of cold redundancies of size N.

Experimental Results. Fig. 11b shows how the state space sizes increase with varying number of redundancies N for both the raw Markov Automaton of the NdDFT and the finally resulting minimized RA. Note that the y-axis is scaled logarithmically. It can be seen that the RA state space grows significantly slower, but still at an exponential pace. However, it can also be seen that the state space reduction remains consistent over the course of the increasing number of redundancies.



(a) Memory system with N redundancies (b) State space growth

Fig. 11. State space growth of RA for memory system with N redundancies

7 Conclusions and Future Work

In this paper, we investigated the problem of optimizing Recovery Automata that represent recovery strategies synthesized from NdDFTs. New algorithms to minimize an RA by additionally eliminating non trace-equivalent states and redundant transitions were provided. In particular, we extended the notion of recovery equivalence between states by introducing the notion of orthogonal states and a rule for merging them. In addition, we introduced the concept of fail states and a rule for merging them with predecessor states. A formal proof showing that an equivalent RA is produced for each case was given. A case study using the described approach was provided and the evaluated results showed that it allows to obtain a more efficient implementation of recovery strategies for FDIR than solely eliminating trace equivalent states.

In the future, we would like to extend the Recovery Automata model to deal with input of Fault Trees with transient and repairable faults and consider how the merging rules can be transferred.

References

1. Beccuti, M., Franceschinis, G., Codetta-Raiteri, D., Haddad, S.: Computing optimal repair strategies by means of NdRFT modeling and analysis. The Computer Journal **57**(12), 1870–1892 (2014). https://doi.org/10.1093/comjnl/bxt134

2. Bittner, B., Bozzano, M., Cimatti, A., De Ferluc, R., Gario, M., Guiotto, A., Yushtein, Y.: An integrated process for FDIR design in aerospace. In: Model-Based Safety and Assessment, LNCS, vol. 8822, pp. 82–95. Springer (2014). https://doi.org/10.1007/978-3-319-12214-4_7
3. Bobbio, A., Portinale, L., Minichino, M., Ciancamerla, E.: Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. Reliability Engineering & System Safety **71**(3), 249–260 (2001). https://doi.org/10.1016/S0951-8320(00)00077-6
4. Codetta-Raiteri, D., Portinale, L.: Dynamic Bayesian networks for fault detection, identification, and recovery in autonomous spacecraft. IEEE Transactions on Systems, Man, and Cybernetics: Systems **45**(1), 13–24 (January 2015). https://doi.org/10.1109/TSMC.2014.2323212
5. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability **41**(3), 363–377 (1992). https://doi.org/10.1109/24.159800
6. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: IEEE Symposium on Logic in Computer Science. pp. 342–351. IEEE (2010). https://doi.org/10.1109/LICS.2010.41
7. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: Quantitative Evaluation of Systems. LNCS, vol. 8054, pp. 55–71. Springer (2013). https://doi.org/10.1007/978-3-642-40196-1_5
8. Hopcroft, J.: An n log n algorithm for minimizing states in a finite automaton. In: Theory of machines and computations, pp. 189–196. Elsevier (1971). https://doi.org/10.1016/B978-0-12-417750-5.50022-1
9. International Electrotechnical Commission, Geneva, Switzerland: Fault Tree Analysis (FTA) (2006)
10. Kildall, G.A.: A unified approach to global program optimization. In: Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages. pp. 194–206. ACM (1973). https://doi.org/10.1145/512927.512945
11. Müller, S., Gerndt, A., Noll, T.: Synthesizing FDIR recovery strategies from non-deterministic dynamic fault trees. In: 2017 AIAA SPACE Forum. vol. AIAA 2017-5163. American Institute of Aeronautics and Astronautics (2017). https://doi.org/10.2514/6.2017-5163
12. Raiteri, D.C., Portinale, L.: Arpha: an fdir architecture for autonomous spacecrafts based on dynamic probabilistic graphical models. Tech. Rep. TR-INF-2010-12-04-UNIPMN, Computer Science Institute, Università del Piemonte Orientale, Vercelli, Italy (December 2010), http://www.di.unipmn.it/TechnicalReports/TR-INF-2010-12-04-UNIPMN.pdf
13. Ruijters, E., Stoelinga, M.: Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. Computer Science Review **15-16**, 29–62 (2015). https://doi.org/10.1016/j.cosrev.2015.03.001
14. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault tree handbook. Tech. rep., Nuclear Regulatory Commission, Washington, DC (1981), https://www.osti.gov/biblio/5762464-fault-tree-handbook
15. Wander, A., Förstner, R.: Innovative fault detection, isolation and recovery strategies on-board spacecraft: State of the art and research challenges. In: Deutscher Luft- und Raumfahrtkongress 2012. German Soc. for Aeronautics and Astronautics – Lilienthal-Oberth e.V., Bonn, Germany (January 2013), https://www.dglr.de/publikationen/2013/281268.pdf